Чарльз Х. Мур (Charles H. Moore) родился в 1938 г.; вырос в штате Мичиган; получил степень бакалавра по физике в MIT; женат на Уинфред Беллис, есть сын Эрик. Сейчас живёт в Инклайн-Виллидж, на чудесном озере Тахо; водит WRX; ходит горными туристическими маршрутами Тахо-Рим и ПасификКрест; много читает. Получает удовольствие, находя простые решения, для чего может изменить задачу.
В 1960-е годы работал как независимый программист, пока не изобрёл в 1968 г. Форт (Forth) - простой, эффективный и гибкий компьютерный язык, которым очень гордится. Занимался программированием телескопов для Национальной радиоастрономической лаборатории (NRAO). В 1971 г. стал сооснователем фирмы Forth, Inc., занимающейся программированием систем реального времени.
В 1983 г., недовольный имевшимся аппаратным обеспечением, стал сооснователем Novix, Inc. Разработал в ней микропроцессор NC4000. Позднее он превратился в Harris RTX2000, который подошёл для использования в космосе и теперь вращается вокруг Сатурна на <Кассини>. В фирме Computer Cowboys с помощью специального программного обеспечения разработал ShBoom, Mup20, F21 and i21 - микропроцессоры с архитектурой Forth. Очень гордится этими маленькими, быстрыми и экономичными процессорами.В этом столетии стал сооснователем IntellaSys и придумал colorForth для программирования конструкторских инструментов для многоядерного чипа. В 2008 г. IntellaSys производила и продавала 40-ядерную версию процессора. В настоящее время Чарльз Мур переносит свои инструменты на этот удивительный чип.

Форт

Форт (Forth) - это стековый, конкатенативный язык, созданный в 1960-х годах Чарльзом Муром.

Его основная особенность:
использование стека для хранения данных и слов для операций, которые берут из стека аргументы и помещают в стек результат.
Язык настолько компактен, что может использоваться как во встраиваемых системах, так и в суперкомпьютерах, и достаточно выразителен, чтобы создавать полезные программы длиной в несколько сотен слов.
Среди продолжателей идеи - colorForth самого Чака Мура (Chuck Moore), а также язык программирования Factor.

Язык Форт и языковое проектирование

Как бы вы определили Форт?

Чак Мур: Форт - компьютерный язык с минимальным синтаксисом. Он характеризуется наличием явного стека параметров, что позволяет эффективно вызывать подпрограммы. Отсюда постфиксные выражения (операторы пишутся после аргументов) и стиль программирования с высокой степенью структурирования, при котором множество коротких программ передают друг другу параметры через стек.

Приходилось читать, что название Forth означало четвёртое поколение ПО.
Не могли бы вы рассказать об этом подробнее?

Чак: <Форт> происходит от fourth (четвёртый), что намекает на <язык программирования четвёртого поколения>. Кажется, я перескочил через поколение. Фортран/Кобол – языки первого поколения, Алгол/ Лисп - второго. Во всех этих языках большую роль играл синтаксис.
Чем сильнее развит синтаксис, тем больше возможностей для проверки ошибок. Однако большая часть ошибок - синтаксические. Я решил минимизировать синтаксис, отдав предпочтение семантике. В самом деле, в Форте слова полны смысла.

Вы рассматриваете Форт как языковый набор инструментов. Можно понимать под этим относительную простоту синтаксиса в сравнении с другими языками программирования и возможность создания словарей из более коротких слов. Или что-то ещё?

Чак: Нет, главное то, что язык обеспечивает высшую степень структурирования.
Программа на Форте состоит из множества коротких слов, тогда как программа на Си состоит из меньшего числа длинных слов. Под коротким словом я понимаю его определение размером примерно в одну строку. Язык строится путём определения

новых слов через уже существующие, и эта иерархия развивается, пока не наберётся, скажем, тысяча слов.
Проблема в том, чтобы:
1) решить,какие слова полезны, и
2) запомнить их все. Я сейчас работаю над приложением, в котором тысяча слов.
И у меня есть инструменты для поиска слов, но искать слово можно, только если знаешь, что оно существует, и примерно помнишь, как оно пишется. Это приводит к особому стилю программирования, и нужно некоторое время, чтобы программист привык так работать.

Я видел множество Форт-программ, которые выглядят, как Си-программы, буквально переведённые на Форт. Но смысл не в этом, а в том, чтобы работать совсем по-другому. Ещё одна интересная особенность этого набора инструментов в том, что всякое новое определённое вами слово столь же эффективно и значимо, как слова, изначально определённые в ядре. Здесь нет никакой дискриминации.

Связана ли такая наглядная структура из множества коротких слов с реализацией Форт?

Чак: Она является результатом очень эффективной схемы вызова подпрограмм. Отсутствует передача параметров, потому что это стековый язык. Есть только вызов подпрограммы и возврат. Стек открыт. Машинный язык компилируется. Вход в подпрограмму и выход из неё - это буквально одна команда call и одна команда return. Кроме того, всегда можно опуститься ниже до эквивалента языка ассемблера. Можно определить слово так, чтобы оно выполняло машинные команды, а не вызывало подпрограммы, поэтому эффективность может быть такой же, как в любом другом языке, а то и повыше.

У вас отсутствуют накладные расходы вызовов Си.

Чак: Верно. Это очень расширяет возможности программиста. Если грамотно структурировать задачу, её решение может быть не только эффективным, но и очень легко читаться. С другой стороны, если сделать это плохо, может получиться код, непонятный никому, - вашему начальнику, например (если он хоть что-то понимает). И можно очень запутать дело. Так что это обоюдоострый меч: может получиться как очень хорошо, так и очень плохо.

Что бы вы сказали (или какой код продемонстрировали) разработчику, использующему другой язык программирования, чтобы вызвать у него интерес к Форту?

Чак: Опытного программиста очень трудно заинтересовать Фортом. Ведь он потратил силы на изучение своего языка/операционной системы и построил свою библиотеку для тех задач, которыми занимается. Рассказ о том, что на Форте всё будет меньше, быстрее и проще, покажется недостаточно убедительным в сравнении с предстоящей необходимостью переписать весь код заново. Начинающий программист или инженер, которому нужно написать код, не сталкивается с таким препятствием и оказывается более благосклонным, как и опытный программист, начинающий работу над новым проектом в новых условиях, например, в моей ситуации -с многоядерными процессорами.

Вы сказали, что многие программы на Форте, которые вы видели, напоминают Си-программы. Как правильно писать программы на Форте?

Чак: Снизу вверх. Начать, видимо, придётся с каких-то сигналов ввода/вывода, которые нужно генерировать, - вот и займитесь ими. Затем напишите код, который управляет генерацией этих сигналов. Потом вы поднимаетесь выше, пока наконец не дойдёте до слова самого верхнего уровня. Вы даёте ему имя go, вводите с клавиатуры go - и понеслось. Я не очень доверяю системным аналитикам, действующим в нисходящем направлении. Они определяют, в чём заключается задача, а потом разбивают её на части так, что реализация бывает очень затруднена.

Проектирование, управляемое предметной областью, требует описать бизнес-логику с помощью лексикона заказчика. Есть ли связь между построением словаря и использованием технических терминов, принятых в предметной области?

Чак: Надо надеяться, что программист ознакомится с предметной областью, прежде чем начнёт писать код. Обычно я беседую с заказчиком. Я слушаю, какие слова он употребляет, и пытаюсь их использовать, чтобы он мог понять, как работает программа. Форт даёт возможность такого облегчённого чтения благодаря своей постфиксной нотации. Если бы я писал финансовое приложение, то, вероятно, в программе было бы слово <percent>. Вы говорите <2.03 percent>, и это 2,03 процента от аргумента, и всё работает и читается самым натуральным образом.

Может ли проект, начатый во времена перфокарт, быть всё ещё полезен для современных компьютеров в эпоху интернета? Форт проектировался на/для IBM 1130 в 1968 году. То, что он оказался превосходным языком для параллельных вычислений в 2007 году, поистине поражает.

Чак: Он развивался всё это время. Но Форт - простейший из возможных компьютерных языков. Он никак не ограничивает программиста. Программист может определять слова, которые кратко отражают характеристики задачи в точном иерархическом виде.

Создавая программу, учитываете ли вы, что она должна хорошо читаться на английском языке?

Чак: Да, если это верхний уровень. Английский плохо подходит для описания функциональности. Он не был для этого предназначен, но у него есть такая же особенность, как у Форта: возможность определять новые слова.
Новые слова определяются в основном путём их объяснения с помощью ранее определённых слов. В естественных языках с этим могут быть проблемы. Если открыть словарь, то можно обнаружить, что определения часто образуют замкнутый круг, и ничего содержательного нельзя извлечь.

Слова вместо синтаксиса с разного вида скобками (как в С): можно ли проявить хороший вкус при написании программ на Форте?

Чак: Надеюсь, что да. Нужно, чтобы программист на Форте заботился о внешнем виде кода, а не только о его функциональности. Приятно, когда соседние слова составляют единый поток. Вот почему я разработал colorForth.
Меня стал раздражать синтаксис Форта. Например, комментарий можно выделить с помощью круглых скобок. Глядя на все эти символы пунктуации, я сказал себе, что должен быть какой-то лучший способ. Этот лучший способ оказался довольно накладным, поскольку к каждому слову исходного кода нужно было прикрепить тег, но когда я справился с возникшими издержками, то с удовольствием увидел, как вместо замысловатых символов появились разноцветные слова - по-моему, гораздо более приятный способ обозначить функциональность. Меня постоянно критикуют те, кто не различает цвета. Они действительно возмущены тем, что я попытался лишить их права быть программистами, но кто-то в конце концов предложил вместо разных цветов использовать разные шрифты, что тоже хороший способ. Главная особенность - 4-разрядный тег для каждого слова, что даёт 16 вариантов действий, и компилятор может сразу определить, что от него требуется, а не догадываться об этом из контекста.

Языки второго и третьего поколений стремились к минимализму, например в реализациях с метациклическим интерпретатором. Форт - яркий пример минимализма в отношении конструкции языка и требуемой аппаратной поддержки. Это веяние времени или ваше достижение?

Чак: Нет, при проектировании намеренно ставилась цель получить минимальное по размеру ядро. Определить как можно меньше исходных слов, и пусть программист сам добавляет новые по мере надобности. Главной причиной была потребность в переносимости. Существовали уже десятки миникомпьютеров, а позднее появились десятки микрокомпьютеров. Я лично устанавливал Форт на многие из них. Я стремился максимально облегчить задачу.
На практике может существовать ядро примерно из сотни слов, которых достаточно, чтобы сгенерировать, так сказать, <операционную систему> (хотя это не совсем точно), в которой будет ещё пара сотен слов. После этого можно писать приложения. Я обеспечивал выполнение первых двух этапов и предоставлял прикладным программистам действовать на третьем, но нередко и сам писал приложения. Я определял слова, которые, по моему мнению, были необходимы.

Первая сотня слов должна быть на машинном языке или ассемблере - во всяком случае, непосредственно взаимодействовать с конкретной платформой. Вторые две-три сотни слов должны быть словами более высокого уровня, минимизирующими машинную зависимость от нижнего, ранее определённого уровня. После этого приложение становится практически машиннонезависимым, и его легко портировать с одного миникомпьютера на другой.

Легко ли вам было выполнять портирование начиная с этого второго уровня?

Чак: В высшей степени. Например, у меня был текстовый редактор, с помощью которого я писал исходный код. Обычно он переносился на другую машину без всяких изменений.

Поговаривают, что всякий раз, столкнувшись с новой машиной, вы немедленно начинали переносить на неё Форт.

Чак: Да. На самом деле, простейшим способом понять, как работает эта машина и каковы её особенности, была оценка лёгкости реализации на ней стандартного набора слов Форта.

Как вы пришли к изобретению косвенного шитого кода?

Чак: Косвенный шитый (indirect-threaded) код - довольно тонкое понятие.
Для каждого слова Форта есть запись в словаре. В косвенном шитом коде каждая запись указывает на код, который нужно выполнить, когда встретится это слово. Косвенный шитый код указывает на место, где находится адрес этого кода.
Это позволяет получить другую информацию помимо адреса - например, значение переменной. Вероятно, это было самое компактное представление слов. Доказано, что оно эквивалентно прямому шитому коду и подпрограммному коду. Конечно, в 1970 году эти понятия и термины не были известны. Но мне показалось, что это самый естественный способ реализовать широкий круг различных слов.

Какое влияние окажет Форт на компьютерные системы будущего?

Чак: Оно уже оказывается. Я уже 25 лет работаю с микропроцессорами, оптимизированными для Форта, и в самое последнее время - с многоядерным процессором, ядра которого представляют собой Форт-компьютеры.
Что даёт Форт? Будучи простым языком, он довольствуется простым компьютером. 256 слов оперативной памяти, 2 стека, 32 команды, асинхронная работа, простая связь с соседями. Небольшой, с малым энергопотреблением.
Форт поощряет структурирование программ. В результате они хорошо удовлетворяют требованиям многоядерных процессоров для параллельной обработки. Большое количество простых программ побуждает тщательно продумать каждую из них.
И требуется при этом какой-нибудь 1% объёма кода,который был бы написан в других случаях. Слыша, как кто-то похваляется миллионами строк кода, я уверен, что этот человек вопиющим образом не разобрался в своей задаче. Нет в наше время задач, которые требовали бы миллионов строк кода. Зато есть небрежные программисты,плохие начальники и неоправданные требования к совместимости.
Запрограммировать на Форте множество небольших компьютеров - превосходное решение. Другие языки не обладают достаточной модульностью или гибкостью. В процессе уменьшения размеров компьютеров и налаживания взаимодействия между их сетями (smart dust - <умная пыль>?) формируется среда будущего.

Это похоже на одну из главных идей UNIX: множество взаимодействующих программ, каждая из которых делаетчто-то одно. Это попрежнему лучшая конструкция? Не лучше ли вместо множества программ на одном компьютере иметь множество программ в сети?

Чак: Идея многопоточного кода, как она реализована в UNIX и в других ОС, была предтечей параллельной обработки. Но между ними есть существенные различия. На большом компьютере можно позволить себе большие накладные расходы, обычно неизбежные при многопоточности. В конце концов, уже существует огромная операционная система. Но для параллельной обработки почти всегда чем больше компьютеров, тем лучше. Когда ресурсы фиксированы, больше компьютеров значит - меньшие компьютеры. А маленькие компьютеры не могут позволить себе такие же накладные расходы, как большие. Маленькие компьютеры должны объединяться - на

одном чипе, на одной плате или беспроводными соединениями. У маленького компьютера маленькая память. Для операционной системы там нет места. Эти компьютеры должны быть автономными, с собственными средствами связи. Поэтому связь должна быть простой - никаких изощрённых протоколов. Программы должны быть компактными и эффективными. Идеальная область для применения Форта. Системы из миллионов строк кода отпадут. Они порождены большими централизованными компьютерами. Для распределённых вычислений нужен иной подход. Язык, созданный для поддержки массивного синтаксического кода, способствует тому, что программисты пишут большие программы. Это поощряется и приносит удовлетворение. Нет факторов, побуждающих стремиться к компактности. С помощью синтаксических языков можно писать компактный код, но обычно этого не происходит. Реализация обобщений, предполагаемых синтаксисом, приводит к неуклюжему и неэффективному объектному коду. Это неприемлемо для маленьких компьютеров. В правильно спроектированном языке исходный код переводится в объектный один к одному. Программисту очевидно, какой код будет сгенерирован из того, что он написал. Это служит источником удовлетворения иного рода, повышает эффективность и сокращает потребность в документации.

Одна из целей создания Форта - достижение компактности как исходного, так и двоичного кода, поэтому данный язык популярен среди программистов встраиваемых приложений. Но программисты во многих других областях могут обоснованно предпочесть другие языки. Есть ли в самом языке особенности, из-за которых исходный или конечный код может только пострадать?

Чак: Форт действительно компактен. Одна из причин - ограниченность синтаксиса. В других языках можно наблюдать намеренное введение синтаксиса, создающего избыточность и возможность синтаксической проверки с целью выявления ошибок. В Форте мало возможностей для обнаружения ошибок ввиду отсутствия избыточности. Благодаря этому исходный код становится компактнее. Мой опыт работы с другими языками показывает, что большинство ошибок связано с синтаксисом. Разработчики языка будто намеренно дают возможность программистам делать ошибки, которые потом может выявить компилятор. Не думаю, что это продуктивно. Это только осложняет и без того трудный процесс написания корректного кода. Примером служит контроль типов. Задание типов для разных чисел позволяет обнаруживать ошибки. Невольным следствием является то, что программист вынужден преобразовывать типы, порой стараясь обойти проверку типа, чтобы достичь своей цели. Другое следствие синтаксиса - необходимость его пригодности во всех предположительных областях применения. В результате синтаксис усложняется. Форт - расширяемый язык. Программист может создавать собственные структуры, столь же эффективные, как предлагаемые компилятором. В результате не нужно заранее предвидеть и обеспечить все возможности, которые могут потребоваться в будущем. Для Форта характерна постфиксная запись. Это упрощает компилятор и обеспечивает однозначную трансляцию исходного кода в объектный. Программист лучше понимает свой код, а код,полученный в результате компиляции, становится компактнее.

Защитники ряда новых языков программирования (в особенности Python и Ruby) указывают на читаемость как на важное преимущество. Насколько легки изучение Форта и поддержка программ на нём в сравнении с этими языками? Что могли бы другие языки позаимствовать у Форта в отношении лёгкости чтения?

Чак: Все компьютерные языки претендуют на лёгкость чтения. Без оснований. Возможно, тому, кто знает язык, кажется, что его легко читать, но новички всегда оказываются в трудном положении. Проблема заключается в загадочном, произвольном и таинственном синтаксисе. Все эти скобки, амперсанды и прочее. Пытаешься выяснить, зачем они там нужны, и в итоге приходишь к выводу: низачем. Но эти правила приходится соблюдать. А говорить на этом языке нельзя. Придётся, как Виктор Борг, произносить все эти символы пунктуации. Форт частично решает эту проблему, минимизируя синтаксис. Его тайные знаки @ и ! читаются как fetch (прочесть) и store (записать). Символы выбраны потому, что операции встречаются очень часто. Программисту рекомендуется использовать слова естественного языка. Они соединяются без всякой пунктуации. Выбирая соответствующие слова, можно составлять разумные фразы. Есть даже стихотворения, написанные на Форте. Другое достоинство - постфиксная нотация. Фраза вроде <6 inches> применяет оператор <inches> (дюймов) к параметру <6>, что очень естественно. Прекрасно читается. С другой стороны, программист должен составить словарь для описания задачи. Словарь может получиться довольно большим. Читателю кода нужно знать этот

словарь, чтобы разобраться в программе. А программист должен постараться определить подходящие слова. Как бы то ни было, чтобы прочесть программу, нужно приложить усилия. На каком бы языке она ни была написана.

Чем определяется успех вашей работы?

Чак: Красивым решением.
На Форте не пишут программы. Сам Форт - это и есть программа. Нужно добавить к нему слова, которые составят словарь для решения задачи. Оно становится очевидным, когда выбраны правильные слова, потому что позволяет интерактивно решать нужные аспекты задачи. Например, я могу определить слова, описывающие электронную схему. Мне понадобится добавить эту схему в чип, показать структуру, проверить выполнение правил, запустить модель. Слова для всего этого составят приложение. Когда они правильно подобраны и составляют компактный и эффективный инструментарий, это и есть для меня успех.

Где вы учились писать компиляторы? Или в те времена этим все занимались по необходимости?

Чак: Я был в Стэнфорде в начале 1960-х, и там была группа старшекурсников, которые писали компилятор Algol для Burroughs 5500. Их, кажется, было всего трое или четверо, и я до глубины души был поражён тем, что три или четыре человека могут сесть и написать компилятор. И я подумал, что раз они могут это сделать, то и я смогу и сделал. Оказалось, что это не так трудно. В те времена компиляторы были окружены неким облаком таинственности.

Это и сейчас так.

Чак: Да, но в меньшей степени. Время от времени появляются всякие новые языки - не знаю уж, компилируемые или интерпретируемые, - и находятся люди хакерского склада, готовые этим заниматься. Другая любопытная вещь - операционная система. Операционные системы чудовищно сложны и совершенно излишни. Билл Гейтс был гениален в том, что ему удалось убедить весь мир в необходимости операционных систем. Пожалуй, в истории человечества другого такого обмана не вспомнить. Операционная система абсолютно не нужна вам. Если у вас есть подпрограмма, назы ваемая драйвером дисков, и другая программа, обеспечивающая поддержку какого-то типа связи, то вам ничего больше не нужно от операционной системы. Windows тратит массу времени, работая с оверлеями, дисками и т.п., что никому не нужно. Сейчас есть гигабайтные диски и мегабайтная оперативная память. Мир изменился, и операционная система стала не нужна.

А как быть с поддержкой устройств?

Чак: Для каждого устройства есть подпрограмма. Нужна библиотека, а не операционная система. Вызовите нужные подпрограммы или загрузите их.

Трудно ли снова программировать после короткого перерыва?

Чак: Не вижу никаких проблем при коротком перерыве. Я сосредоточиваю всё своё внимание на задаче, думая о ней днём и ночью. Наверно, это особенность Форта: полная отдача сил за короткое время (дни), пока задача не будет решена.
Хорошо, что приложения Форта естественным образом распадаются на подпроекты. Обычно код на Форте прост и легко читается. Делая что-то действительно замысловатое, я пишу пространные комментарии. Хорошие комментарии помогают освежить в памяти задачу, но всегда нужно прочесть и понять код.

Какую самую крупную ошибку вы совершили в области программирования или проектирования? Какие уроки вы извлекли из неё?

Чак: Лет 20 назад я хотел создать инструмент для проектирования СБИС. На моём новом ПК не было Форта, поэтому я решил попробовать другой подход: с помощью машинного языка. Не ассемблера, а реальных шестнадцатеричных команд.
Я создал код так, как делал это в Форте: множество простых слов, иерархически взаимодействующих между собой. Получилось. Я пользовался им в течение десяти лет. Но с сопровождением и документацией были сложности. В конце концов я переписал программу на Форте, и она стала меньше и проще. Я сделал вывод, что

Форт эффективнее машинного языка. Отчасти благодаря интерактивности, отчасти - синтаксису. Удобная особенность кода на Форте - документирование чисел с помощью выражений, применяемых для их расчёта.

Аппаратное обеспечение

Как следует разработчику рассматривать аппаратную платформу, на которой он работает, - как ресурс или как ограничение? Если как ресурс, то может возникнуть желание оптимизировать код, используя все возможности, предоставляемые аппаратурой; если как ограничение, то можно писать код в расчёте на то, что с новым, более мощным аппаратным обеспечением он будет работать лучше, а при современных темпах прогресса этого не придётся долго ждать.

Чак: Это очень верное наблюдение - что всякая программа рассчитана на определённое аппаратное обеспечение. Программы для ПК, несомненно, рассчитаны на появление более быстрых компьютеров, и потому им дозволена небрежность.
Но программы для встроенных систем предполагают сохранение аппаратуры в течение всего срока существования проекта. Из одного проекта в другой переходит лишь небольшая часть программного обеспечения. Поэтому аппаратная часть здесь ограничивает, но не абсолютно. В то же время для ПК аппаратное обеспечение является развивающимся ресурсом. Переход к параллельной обработке может всё изменить. Приложения, не способные выполняться на нескольких компьютерах, будут ограниченными, поскольку скорость отдельных компьютеров перестанет расти. Переделывать старые программы, чтобы оптимизировать их параллельное выполнение, невыгодно. А расчёт на то, что положение спасут умные компьютеры, - бесплодные мечтания.

В чём главная проблема параллелизма?

Чак: Главная проблема параллельной обработки в скорости. Компьютер должен решать в приложении множество задач. Это можно осуществить на одном процессоре с помощью многозадачности. А можно одновременно выполнять задачи на нескольких процессорах. Последнее осуществляется гораздо быстрее, и современное программное обеспечение нуждается в такой скорости.

Где кроется решение - в аппаратном обеспечении, в программном или в их сочетании?

Чак: Соединить несколько процессоров нетрудно. Таким образом, аппаратное обеспечение есть. Если программное обеспечение написано в расчёте на него, то проблема решена. Но если можно переписать программу заново, то можно сделать её настолько эффективной, что многопроцессорность не понадобится. Проблема в том, чтобы использовать многопроцессорные системы для старых программ, не меняя их. Задача создания таких интеллектуальных компиляторов не решена. Меня поражает, что написанные в 1970-х программы нельзя переписать. Одна из причин может быть связана с тем, что в ту эпоху программирование было очень увлекательным: всё делалось впервые, и программисты работали по 18 часов в день ради своего удовольствия. Сейчас программируют с 9 до 5 часов, в составе команды и по графику работы - удовольствия мало. Поэтому пытаются добавить ещё один уровень программного обеспечения, чтобы избежать переписывания старых программ. Во всяком случае, это интереснее, чем заново писать дурацкий текстовый редактор.

Современные компьютеры предоставили нам большие вычислительные мощности, но в какой степени эти системы действительно занимаются вычислениями? А в какой - просто перемещают данные и форматируют их?

Чак: Вы совершенно правы. По большей части компьютеры заняты перемещением данных, а не расчётами. Не просто перемещением, но сжатием и шифрованием. При больших скоростях передачи данных этим должны заниматься специальные схемы, поэтому непонятно, зачем вообще нужны компьютеры.

Следует ли что-то из этого? Может быть, нам нужно другое аппаратное обеспечение? Дональд Кнут однажды предложил: посмотрите, чем занимается ваш компьютер в течение отдельно взятой секунды. По его словам, результаты такого исследования могут значительно из менить наши представления.

Чак: В моём компьютере это учтено, и умножение выполняется просто и медленно. Оно используется нечасто. Зато важно, как происходит обмен данными между ядрами и с памятью.

С одной стороны - язык, дающий реальную возможность создавать свои словари, не слишком заботясь об аппаратной части. С другой стороны - очень маленькое ядро, тесно связанное с конкретной аппаратурой. Любопытно, каким образом Форт преодолевает этот разрыв.Правда ли, что на некоторых машинах у вас нет никакой операционной системы, кроме ядра Форта?

Чак: Форт совершенно автономен.

Всё, что нужно, есть в ядре. Но это абстрагирует аппаратную часть для программистов на Форте.

Чак: Верно.

Что-то похожее делала машина Лиспа, но она никогда не стала популярной. Форт незаметно сделал то же самое.

Чак: Лисп не занимался вводом/выводом. Надо сказать, что Си тоже не занимался вводом/выводом, поэтому для него потребовалась операционная система. Форт занимался вводом/выводом с самого начала. Я не верю в наименьшее общее кратное. Я думаю, что когда вы берётесь за новую машину, она потому и новая, что делает что-то иначе, и вы хотите воспользоваться её особенностями. Вам придётся спуститься до уровня ввода/вывода, чтобы суметь это сделать.

Керниган и Ричи могли бы возразить, что для упрощения портирования им нужен был такой Си, который стал бы наименьшим общим кратным. Вы решили,что упростить портирование можно, не придерживаясь такого подхода.

Чак: Я подходил к этому стандартно. У меня было слово - кажется, fetchp, - которое получало 8 бит из порта. На разных компьютерах оно определялось поразному, но в стеке оказывалась одна и та же функция.

Тогда в некотором смысле Форт эквивалентен Си со стандартной библиотекой ввода/вывода.

Чак: Да, но когда-то давно я работал со стандартной библиотекой Фортрана, и это было ужасно. Слова были совершенно не те. Она была очень громоздкой, неэкономной. Можно было с лёгкостью избавиться от лишней нагрузки готового протокола, определив полдюжины команд для операций ввода/вывода.

И часто вы занимались такими обходными путями?

Чак: С Фортраном - да. Когда имеешь дело, скажем, с Windows, деваться некуда. Система не подпустит тебя к вводу/выводу. Я почти намеренно сторонился Windows, но и без Windows процессор Pentium оказался самым сложным устройством для размещения Форта. У него слишком много инструкций. И слишком много аппаратных функций вроде буферов ассоциативной трансляции и разного рода кэшей, которыми нельзя пренебречь. Пришлось с ними разбираться, и код инициализации для запуска Форта оказался очень сложным и громоздким. Хотя он должен был выполняться единственный раз, мне всё же пришлось потратить много времени, чтобы это делалось правильно. Но мы запустили на Pentium автономный Форт, так что труд был не напрасен. Эта работа затянулась, наверное, лет на десять, отчасти потому, что пришлось гнаться за модификациями, которые делала Intel.

Вы сказали, что Форт реально поддерживает асинхронную работу. В каком смысле вы говорили об асинхронной работе?

Чак: В нескольких сразу. В Форте всегда была возможность мультипрограммности и многопоточности - средство под названием Cooperative. Есть слово pause. Если некоторая задача доходит до такого места, где ей не нужно немедленно что-то делать, она должна сказать pause. Циклический планировщик даст компьютеру следующую задачу для выполнения. Если не сказать pause, можно полностью

монополизировать компьютер, но этого не произойдёт, потому что это специальный компьютер. На нём выполняется единственное приложение, и все его задачи дружественны. Думаю, в прежние времена все задачи были дружественны. Это один вид асинхронности, когда все эти задачи могли выполняться и заниматься своими делами без необходимости синхронизации. Опять-таки, одна из особенностей Форта состоит в том, что слово pause могло находиться в словах нижнего уровня. При каждой попытке чтения или записи диска слово pause выполнялось бы безвашего участия, потому что разработчики кода для работы с диском знали, что придётся подождать конца выполнения операции. В новых чипах - тех многоядерных чипах, которые я разрабатываю, - мы придерживаемся той же философии. Каждый компьютер работает независимо, и если на одном компьютере выполняется одна задача, а на соседнем - другая, они выполняются одновременно, но обмениваются одна с другой данными. Так же ведут себя задачи на многопоточном компьютере. Форт лишь удачно выделяет эти независимые задачи. На самом деле, если это многоядерный компьютер, я, возможно, буду использовать немного другие программы, но я могу точно так же структурировать их для параллельного выполнения.

Был ли при кооперативной многопоточности у каждого потока свой стек и происходило ли переключение между ними?

Чак: При переключении задач иногда всё, что требовалось (в зависимости от компьютера), это поместить слово на вершину стека и после этого переключить указатель стека. Иногда, действительно, приходилось копировать стек и загружать новый, но тогда я старался делать стек очень неглубоким.

Вы намеренно ограничивали глубину стека?

Чак: Да. Сначала стеки были произвольной глубины. У первого проектируемого мной чипа глубина стека была 256, потому что я боялся, что её не хватит. У другого спроектированного мной чипа был стек глубиной 4. Сейчас я остановился на том, что глубина стека должна быть около 8 или 10, так что мой минимализм стал строже.

Я бы предположил, что тенденция была противоположной.

Чак: В моём приложении для проектирования СБИС действительно есть случай рекурсивной трассировки чипа, где мне пришлось увеличить глубину стека примерно до 4000. Для этого мог бы потребоваться стек другого типа – программно реализованный. Но на Pentium такой стек можно сделать и аппаратным.

Разработка приложений

Вы высказали мысль, что Форт - идеальный язык для объединения в сеть множества малых компьютеров, как в <умной пыли>, например. Как вы считаете, для каких приложений могли бы с успехом использоваться такие малые компьютеры?

Чак: Несомненно, в области связи, и несомненно, в сборе данных. Но я только начинаю разбираться, как независимые компьютеры могут кооперироваться для выполнения сложных задач. Многоядерные компьютеры, которые у нас есть, чрезвычайно малы. У них 64 слова памяти. Хотя можно считать, и что 128 слов: 64 RAM, 64 ROM. В каждом слове может храниться до четырёх инструкций. В итоге у вас окажется в каждом компьютере не более 512 инструкций, поэтому задача должна быть достаточно простой. Спрашивается, если взять такую задачу, как, скажем, стек TCP/IP, то как разбросать её по множеству таких компьютеров, чтобы выполнялась нужная функция и при этом на каждом из компьютеров было не более 512 команд? Очень красивая проблема, за которую я как раз взялся. Думаю, это касается почти любых приложений. Приложению гораздо проще работать, если оно разбито на независимые части, а не пытается делать то же самое последовательно на одном процессоре. Полагаю, это относится к генерации видео. Наверняка это справедливо для компрессии и декомпрессии изображений. Но я ещё только учусь это делать. У нас в компании есть и другие люди, которые тоже учатся и получают от этого удовольствие.

Есть какая-то область применения, где этот подход не годится?

Чак: Конечно - для старых программ. Меня реально тревожит устаревшее программное обеспечение, и если вы хотите вернуться к этой проблеме, пожалуй, более естественно описать её следующим образом. Мне кажется, что она тесно связана с тем, как мы представляем работу мозга с помощью независимых агентов Мински. Агент представляется мне маленьким ядром. Вполне возможно, что разум возникает в процессе взаимодействия между ними, а не как функция отдельного агента. Старые программы - это недооценённая, но серьёзная проблема. Со временем она будет лишь усугубляться - не только в банковской сфере, но и в аэрокосмической области и других отраслях. Проблема в том, что это миллионы строк кода. Их можно переписать заново, и это будут, скажем, тысячи строк на Форте. В машинном переводе смысла нет - это только увеличит размер кода. Но этот код невозможно проверить. Стоимость и риски будут ужасающими. Устаревший код может привести нашу цивилизацию к гибели.

Похоже, вы совершенно уверены в том, что в ближайшие 10-20 лет мы всё чаще будем встречаться с тем, что программы образуются в результате свободного соединения множества мелких частей.

Чак: О да. Я уверен, что так и будет. Радиосвязь - чудесная вещь. Говорят, что в тело человека будут внедряться микроагенты, одни из которых будут служить датчиками, а другие - исполнительными механизмами, но связываться между собой они смогут только с помощью радио или звуковых волн. Их возможности будут ограниченны - это всего несколько молекул. Поэтому это будет похоже на окружающий мир. Так организовано наше человеческое сообщество. Оно состоит из шести с половиной миллиардов независимых агентов, которые взаимодействуют между собой.

Плохой выбор слов может привести к плохо спроектированным и трудно сопровождаемым приложениям. Не приводит ли создание крупного приложения из десятков и сотен мелких слов к появлению тарабарщины? Как этого избежать?

Чак: Практически никак. Я сам иногда обнаруживаю, что плохо подбираю слова. А тогда можно совсем запутаться. Помню, как в одном приложении я выбрал слово - сейчас уже забыл, какое - и потом изменил его, так что в итоге оно имело смысл, прямо противоположный звучанию. Как если бы слово вправо сдвигало всё влево. Это было ужасно. Какое-то время я боролся с этим, а потом переименовал слово, потому что совершенно невозможно было понять программу, пока это слово так затуманивало смысл. Я предпочитаю использовать английские слова, а не сокращения. Люблю писать их целиком. С другой стороны, я хочу, чтобы они были короткими. Короткие и осмысленные английские слова быстро кончаются, и приходится придумывать что-то другое. Терпеть не могу префиксы; это грубый способ создания пространств имён, чтобы многократно использовать одни и те же слова. Они мне кажутся отговоркой. Так легко различать слова, но неужели нельзя было придумать что-то поумнее! Очень часто в приложениях на Форте есть отдельные словари, в которых можно повторно использовать слова. В одном контексте слово означает одно, в другом - другое. Когда я проектировал СБИС, от этих идей пришлось отказаться. Мне нужно было не меньше тысячи слов, и не просто английские слова, а названия сигналов и другое в этом роде, и мне быстро пришлось вернуться к определениям и словам со странным правописанием, а также к префиксам и подобным вещам. Такой код трудно читать. С другой стороны, там много слов вроде nand, или nor, или xor, обозначающих разные вентили. Я использую слова, где только можно. Я вижу, как люди программируют на Форте. Не хочу сказать, что только я умею правильно программировать на Форте. Некоторые умеют придумывать очень хорошие имена, другие справляются с этим очень плохо. Одни создают очень хороший синтаксис, другие не придают этому значения. У одних получаются очень короткие определения слов, У других слово занимает полстраницы. Нет правил - есть только стилистические договорённости. Кроме того, у Форта есть важное отличие от Си, Пролога, Алгола и Фортрана, связанное с тем, что обычные языки старались предусмотреть все мыслимые структуры и синтаксис, встроив их в язык. Из-за этого языки получались очень неуклюжими. Я считаю, что Си - очень неизящный язык со всеми своими квадратными и фигурными скобками, двоеточиями и точками с запятой и всем прочим. В Форте всё это отсутствует. Я не ставил целью решение общей задачи. Я лишь собирался сделать инструмент, с помощью которого кто-то другой мог бы решить любую задачу, вставшую перед ним. Возможность сделать любую вещь, а не возможность делать все вещи.

Нужно ли поставлять вместе с микрокомпьютерами исходный код, чтобы их можно было исправить даже десять лет спустя?

Чак: Верно, исходный код составил бы прекрасную документацию для микрокомпьютеров. Компактность Форта способствовала бы этому. Но дальше придётся также включить компилятор и редактор, чтобы можно было посмотреть и модифицировать код микрокомпьютера без помощи другого компьютера или операционной системы, которые могут быть утеряны. colorForth и представляет мою попытку сделать это. Всё, что требуется, это несколько килобайт исходного и/или объектного кода. Их легко записать во флеш-память и использовать в будущем.

Как связаны между собой конструкция языка и конструкция программ на нём?

Чак: Применение определяется его особенностями. Это видно на примере языков общения между людьми. Взгляните на различия между романскими (французский, итальянский), западными (английский, немецкий, русский) и восточными (арабский, китайский) языками. Они влияют на соответствующие культуры и мировоззрения. Из всех этих языков особой краткостью выделяется английский, и его распространённость всё время растёт. То же касается языков для общения человека с машиной. Первые языки (Кобол, Фортран) были очень многословны. В последующих языках (Алгол, Си) был избыточный синтаксис. Такие языки неизбежно приводили к тому, что описания алгоритмов были большими и неизящными. На них можно было описать всё, что угодно, но плохо. Форт решает эти проблемы. Он в значительной мере лишён синтаксиса. Он способствует написанию коротких и эффективных определений. Он минимизирует потребность в комментариях, которые часто бывают неточными и отвлекают внимание от собственно кода. В Форте также организован простой и эффективный вызов подпрограмм. Вызов подпрограмм в Си требует дорогостоящей подготовки и последующего восстановления. Это отталкивает от его употребления. Способствует созданию сложных наборов параметров, чтобы окупить стоимость вызова, но приводит к большим и сложным подпрограммам. Эффективность вызова допускает разбиение программ Форт на множество мелких подпрограмм, что обычно и происходит. Для моего стиля характерны однострочные определения - сотни мелких подпрограмм. В этом случае выбор используемых в коде имён приобретает большое значение - как для мнемоники, так и для читаемости программы. Когда код легко читается, меньше потребность в документации. Благодаря отсутствию в Форте синтаксиса можно позволить себе вольности. Для меня это означает возможность проявить творчество и писать код, на который приятно смотреть. Некоторые рассматривают это как недостаток, мешающий контролю со стороны руководства и стандартизации. Думаю, вина в таких случаях лежит на руководстве, а не на языке.

Вы сказали, что большинство ошибок возникает из-за синтаксиса. Как избежать в программах на Форте других ошибок, например связанных с логикой, сопровождением и выбором плохого стиля?

Чак: Основные ошибки в Форте совершаются при работе со стеком. Можно случайно что-нибудь оставить в стеке и потом поскользнуться на этом. Очень важно связывать со словами комментарии к стеку. Нужно описать, что находится в стеке при входе и что - при выходе. Но это всего лишь комментарии. Полностью доверять им нельзя. Иногда их реально выполняют для верификации и выяснения состояния стека. В принципе, выход в разбиении. Если определение вашего слова занимает одну строку, можно прочесть его до конца, проследив поведение стека и проверив, что всё правильно. Можно протестировать слово и убедиться, что оно работает так, как вы задумали, но при этом допустить ошибку со стеком. Слова dup и drop встречаются постоянно, и ими нужно корректно пользоваться. Очень большое значение имеет возможность выполнять слова вне контекста, просто задав для них входные параметры и посмотрев на выходные. Опять-таки, действуя в восходящем порядке, вы можете быть уверены, что все ранее определённые вами слова работают правильно, потому что вы их протестировали. Кроме того, в Форте мало условных операторов. Есть конструкции if-else-then и begin-while. Моя философия, которую я стараюсь систематически распространять, состоит в том, что количество условных операторов в программе должно быть минимально. Вместо одного слова, которое проверяет некое условие, а потом выполняет либо одно, либо другое, лучше иметь два слова: одно выполняет это, другое - то, а вы решаете, которое из них применить. В Си так не получится, потому что вызов обходится дорого, и

стараются ввести такие параметры, чтобы одна и та же подпрограмма выполняла разные действия в зависимости от того, как её вызвали. Отсюда все ошибки и осложнения в старых программах.

В попытках обойти недостатки реализации?

Чак: Да. От циклов не уйти. Циклы могут быть весьма и весьма полезны. Но в Форте, во всяком случае, в colorForth, циклы очень просты: у них один вход и один выход.

Что вы посоветуете новичкам, чтобы они могли программировать эффективно и с удовольствием?

Чак: Я наверняка не удивлю вас, сказав, что нужно учиться писать код на Форте. Даже если вы не собираетесь профессионально писать программы на Форте, поработав с ним, вы извлечёте некоторые уроки, которые пригодятся вам, с каким бы языком вы ни стали работать. Если бы я писал программу на Си - а я почти не писал их, - то стал бы делать это в стиле Форта, с множеством простых подпрограмм. Даже если это потребуетбольше труда, вы облегчите поддержку. Другой принцип - простота. Чем бы вы ни занимались - проектированием самолёта или написанием программы, даже обычного текстового редактора, вы неизбежно начинаете добавлять всё новые и новые функции, пока цена не станет неприемлемой. Лучше сделать несколько текстовых процессоров, ориентированных на разные рынки. Глупо писать электронное письмо с помощью Word: 99% его возможностей окажутся невостребованными. Для электронной почты нужен свой редактор. Когда-то такой был, но мода идёт в другом направлении. Непонятно почемую. Будьте проще. Если вы разрабатываете приложение, если входите в команду проектировщиков, постарайтесь убедить остальных, что нужно стремиться к простоте. Не нужно прогнозировать. Не нужно решать задачу, которая, как вам кажется, может возникнуть в будущем. Решайте ту задачу, которая стоит сейчас. Стремление предугадать неэффективно. Вы рассчитываете на 10 событий, из которых в действительности случится одно, в результате кучу сил потратите впустую.

Каковы признаки простоты?

Чак: Мне кажется, что наука о сложности только зарождается, и одна из её догм - измерение сложности. Мне нравится определение - не знаю, есть ли другие, - что из двух концепций проще та, у которой короче описание. Если вы описали нечто короче, значит, описали его проще. Но здесь скрыт подвох, поскольку любое описание зависит от контекста. Вы напишете очень короткую программу на Си и будете считать, что она очень проста, но в действительности вы опираетесь на то, что есть компилятор Си, операционная система и компьютер, где всё это будет выполняться. Поэтому в более широком контексте у вас окажется не простая, а довольно сложная вещь. Я бы сравнил это с красотой. Невозможно определить её, но, видя красоту, вы узнаёте её; простое - значит маленькое.

Как работа в команде влияет на программирование?

Чак: Групповую работу слишком превозносят. Первая задача группы - разбить задачу на относительно независимые части. Назначьте исполнителя для каждой части. Руководитель команды отвечает за стыковку отдельных частей между собой. Иногда могут работать вместе два человека. Обсуждая задачу, они делают её более понятной. Но интенсивный обмен информацией становится самоцелью. Групповое мышление не способствует творчеству. И когда несколько человек работают вместе, наверняка всю работу делает один.

Это происходит в любом проекте? Если требуется богатая функциональность, как в OpenOffice.org... ведь это достаточно сложно?

Чак: Такие вещи, как OpenOffice.org, разбиваются на подпроекты, каждый из которых программируется одним человеком, а общение поддерживается на уровне, необходимом для обеспечения совместимости.

По каким признакам вы узнаёте хорошего программиста?

Чак: Хороший программист быстро пишет хороший код - корректный, компактный и читаемый. <Быстро> значит несколько часов или дней. Плохому программисту нужно обсудить задачу, и он будет тратить время на планирование, вместо того чтобы писать код, и сделает своей профессией написание и отладку кода.

Что вы думаете о компиляторах? Некажется ли вам, что они позволяют скрыть уровень мастерства программиста?

Чак: Компиляторы, пожалуй, худшие образцы кода. Их пишут те, кто никогда не писал компиляторов раньше и никогда не будет делать этого впоследствии. Чем изощрённее язык, тем более сложным, изобилующим ошибками и непригодным для работы оказывается компилятор. Но простой компилятор для простого языка - важный инструмент, хотя бы для документации. Редактор важнее компилятора. Большое разнообразие редакторов позволяет каждому программисту выбрать какой-то свой, что мешает успешному сотрудничеству. На этой основе процветает производство доморощенных трансляторов из одного формата в другой. Другая беда авторов компиляторов - стремление использовать все специальные символы, имеющиеся на клавиатуре. Так клавиатуры никогда не станут проще и меньше. А исходный код становится непостижимым. Но мастерство программиста не зависит от этих инструментов. Он может быстро освоиться со всеми их недостатками и начать писать хороший код.

Какой должна быть программная документация?

Чак: Я меньше придаю значения комментариям, чем это обычно принято.
Причины:
• Краткие комментарии обычно малопонятны. Приходится догадываться об их смысле.
• Пространные комментарии забивают код, в который помещены для пояснения. Бывает трудно найти код, связанный с комментарием.
• Часто комментарии плохо написаны. Программисты обычно не отличаются литературным талантом, особенно если английский не является для них родным языком. Жаргон и грамматические ошибки часто делают комментарии нечитаемыми.
• Самое главное, комментарии часто оказываются неточными. Код может измениться, а в комментариях это не отражается. Комментарии реже подвергаются критическому анализу, чем код. Неточные комментарии чреваты большими неприятностями, чем их отсутствие. Читателю приходится самому выяснять, где правда - в коде или в комментариях. Комментарии часто бывают не по делу. Они должны пояснять цель кода, а не сам код. Излагать код другими словами бесполезно. А если ещё и неточно, то это просто вводит в заблуждение. Комментарии должны рассказывать, для чего нужен этот код, что он должен делать и какими приёмами это достигается.

В colorForth комментарии выделяются в затенённый блок. Это отделяет их от кода и облегчает чтение. В то же время они легко доступны для чтения и обновления. Кроме того, размер комментариев при этом ограничивается размерами кода.
Комментарии не заменяют настоящую документацию. Для каждого модуля кода нужно написать поясняющий его текстовый документ. Он должен быть значительно подробнее комментариев и давать грамотное и полное объяснение.
Конечно, это делается редко, и не всегда для этого есть возможности, да и потерять документ, который отделён от кода, достаточно легко.
Цитирую по http://www.colorforth.com/HOPL.html
<Вопрос патентования Форта долго обсуждался. Но поскольку патенты на программы являются спорными и могут потребовать участия Верховного суда, NRAO отказалась от преследования этой цели. По этой причине права вернулись ко мне. Глядя назад, можно признать, что единственным шансом для Форта был общественный доступ. Так он и достиг расцвета>.

Патенты на программы и сейчас вызывают споры. Осталось ли прежним ваше мнение о патентах?

Чак: Я никогда не был сторонником патентов на программы. Это похоже на патентование мыслей. А патентование языка/протокола особенно сомнительно. Язык может стать удачным только тогда, когда им пользуются. Всё, что ограничивает его распространение, - глупость.

Вы считаете, что патентование технологии сдерживает или ограничивает её распространение?

Чак: Торговать программами трудно, потому что их легко копировать. Компании изовсех сил стараются защищать свои продукты, в результате чего они иногда делаются непригодными для работы. Я считаю, что продавать нужно <железо>, а программы отдавать бесплатно. Аппаратную часть трудно копировать, а с разработанным для неё программным обеспечением она становится дороже.

Патенты - один из способов решения этих проблем. Они доказали свою плодотворность для развития инноваций. Но необходимо соблюдать тонкую меру, отвергая необоснованные патенты и поддерживая преемственность с прежними методами/патентами. Кроме того, выдача патентов и их реализация сопряжены с огромными расходами. Недавние предложения по реформированию патентного законодательства несут в себе угрозу подавить личное изобретательство, отдав предпочтение большим компаниям. Это стало бы трагедией.